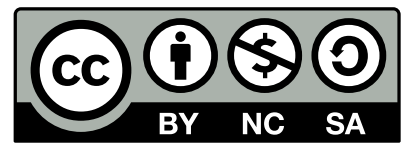
Git Intermediate

Gilles Coremans

ULYSSIS

26 November 2020

This work is licensed under a Creative Commons "Attribution-NonCommercial-ShareAlike 4.0 International" license.



Commit messages

ULYSSIS

Commit houden je geschiedenis overzichtelijk!

Eerste regel: *Korte* beschrijving van changes in imperatief.

Tweede regel: Altijd leeg.

Body: Langere beschrijving van **waarom** en **wat**, zo gedetailleerd als gewenst.

Commit messages: voorbeeld


```
Run more services as root for user namespaces

When Elastic's containers are running in a docker
instance using user namespaces, some crashes can
occur.  Although not consistent, we can mitigate the
issues by running these specific services as root.
```

Working directory & staging area

Working directory: De files op de disk

Staging area/index: De changes die in de volgende commit komen, of de staat zoals ze in de volgende commit komt

Repository: Commits, changes die definitief in Git zitten

git reset

`git reset <commit> -- <file>` haalt `file` uit `commit` naar de index.

`git reset <commit>` verplaatst de branch pointer naar `commit`.

Er zijn 3 soorten `git reset`:

git reset

ULYSSIS

`git reset <commit> -- <file>` haalt `file` uit `commit` naar de index.

`git reset <commit>` verplaatst de branch pointer naar `commit`.

Er zijn 3 soorten `git reset`:

`git reset --soft` Verplaatst enkel de branch pointer

- Behoudt de working tree
- Index is **huidige commit**, niet enkel changes!

git reset

ULYSSIS

`git reset <commit> -- <file>` haalt `file` uit `commit` naar de index.

`git reset <commit>` verplaatst de branch pointer naar `commit`.

Er zijn 3 soorten `git reset`:

`git reset --soft` Verplaatst enkel de branch pointer

- Behoudt de working tree
- Index is **huidige commit**, niet enkel changes!

`git reset --mixed` Behoudt de working tree maar reset de index.

git reset

`git reset <commit> -- <file>` haalt `file` uit `commit` naar de index.

`git reset <commit>` verplaatst de branch pointer naar `commit`.

Er zijn 3 soorten `git reset`:

`git reset --soft` Verplaatst enkel de branch pointer

- Behoudt de working tree
- Index is **huidige commit**, niet enkel changes!

`git reset --mixed` Behoudt de working tree maar reset de index.

`git reset --hard` Reset alles naar de gegeven commit.

- Dit **verwijdert changes permanent!**

Tijdreizen met `git reset`

`git reset` verplaatst de branch pointer, en kan dus **geschiedenis herschrijven!**

Tijdreizen met `git reset`

`git reset` verplaatst de branch pointer, en kan dus **geschiedenis herschrijven!**

`git reset --hard HEAD --` Verwijder alle uncommitted changes

Tijdreizen met `git reset`

`git reset` verplaatst de branch pointer, en kan dus **geschiedenis herschrijven!**

`git reset --hard HEAD --` Verwijder alle uncommitted changes

`git reset --hard HEAD~1 --` Verwijder uncommitted changes **en** vorige commit

Tijdreizen met `git reset`

`git reset` verplaatst de branch pointer, en kan dus **geschiedenis herschrijven!**

`git reset --hard HEAD --` Verwijder alle uncommitted changes

`git reset --hard HEAD~1 --` Verwijder uncommitted changes **en** vorige commit

`git reset --soft HEAD~1 --` Verwijder vorige commit *maar behoud alle changes in die commit*

Tijdreizen met `git reset`

ULYSSIS

`git reset` verplaatst de branch pointer, en kan dus **geschiedenis herschrijven!**

`git reset --hard HEAD --` Verwijder alle uncommitted changes

`git reset --hard HEAD~1 --` Verwijder uncommitted changes **en** vorige commit

`git reset --soft HEAD~1 --` Verwijder vorige commit *maar behoud alle changes in die commit*

`git commit --amend` Pas de vorige commit aan

Tijdreizen met `git rebase`

`git rebase --interactive HEAD~3` past de 3 vorige commits aan:

- **pick** Behoud de commit zoals hij is.
- **squash** Voeg samen met vorige commit.
- **edit** Stop na deze commit om te amenden, nieuwe commits te maken, ...
- **drop** Verwijder de commit.

En veel andere opties!

Je kan ook de volgorde van commits veranderen!

git cherry-pick

`git cherry-pick <commit>` haalt de changes in `commit` naar de huidige branch.

- Als een nieuwe commit: andere parent, dus andere hash.

Werkt ook met commit ranges: `git cherry-pick <to>..<from>`

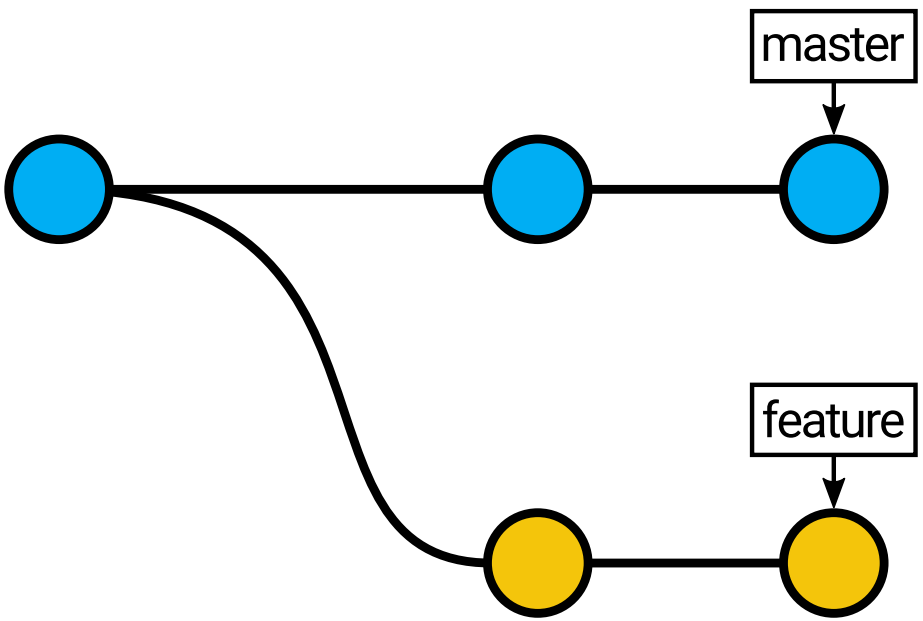
Branches rebasen

`git rebase <upstream> <branch>`: Verplaats het *begin* van de `branch` branch naar de *tip* van `upstream`

Branches rebasen

ULYSSIS

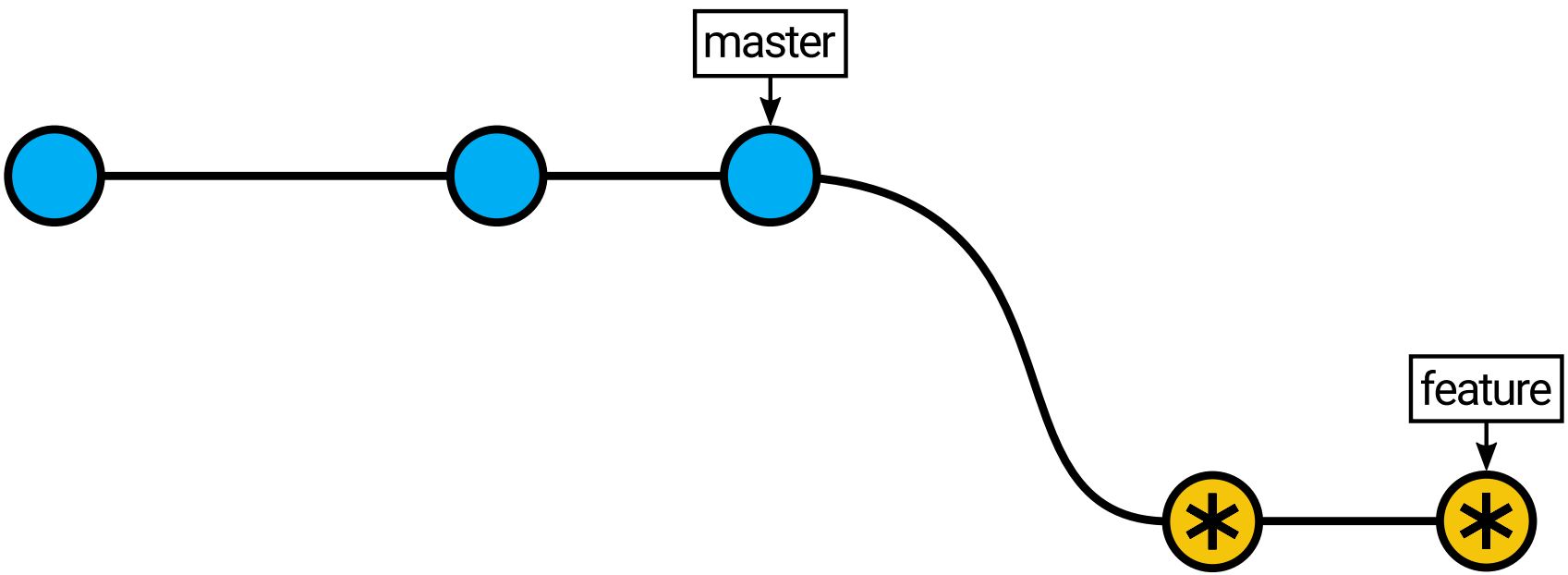
`git rebase <upstream> <branch>`: Verplaats het *begin* van de `branch` branch naar de *tip* van `upstream`



Branches rebasen

ULYSSIS

`git rebase <upstream> <branch>`: Verplaats het *begin* van de `branch` branch naar de *tip* van `upstream`



Waarom rebasen?

`git merge`:

- Merge commits

`git rebase`:

- Geen merge commits

Waarom rebasen?

`git merge`:

- Merge commits
- Behoudt geschiedenis

`git rebase`:

- Geen merge commits
- Herschrijft geschiedenis

Waarom rebasen?

`git merge`:

- Merge commits
- Behoudt geschiedenis
- Resulteert in complexe geschiedenis

`git rebase`:

- Geen merge commits
- Herschrijft geschiedenis
- Resulteert in simpele, lineaire geschiedenis

Waarom rebasen?

`git merge`:

- Merge commits
- Behoudt geschiedenis
- Resulteert in complexe geschiedenis

`git rebase`:

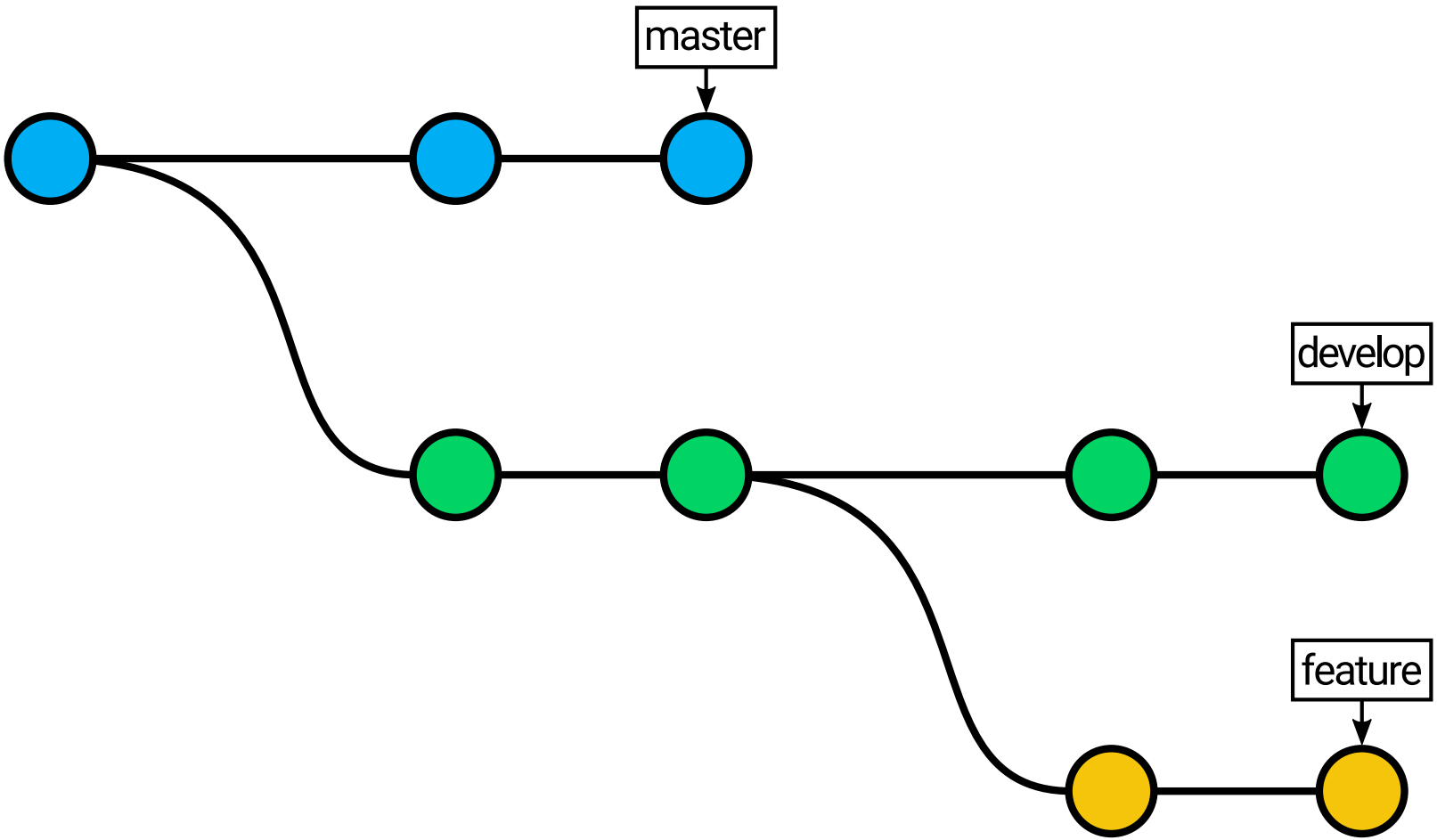
- Geen merge commits
- Herschrijft geschiedenis
- Resulteert in simpele, lineaire geschiedenis

Combineer ze!

Complexere rebase

ULYSSIS

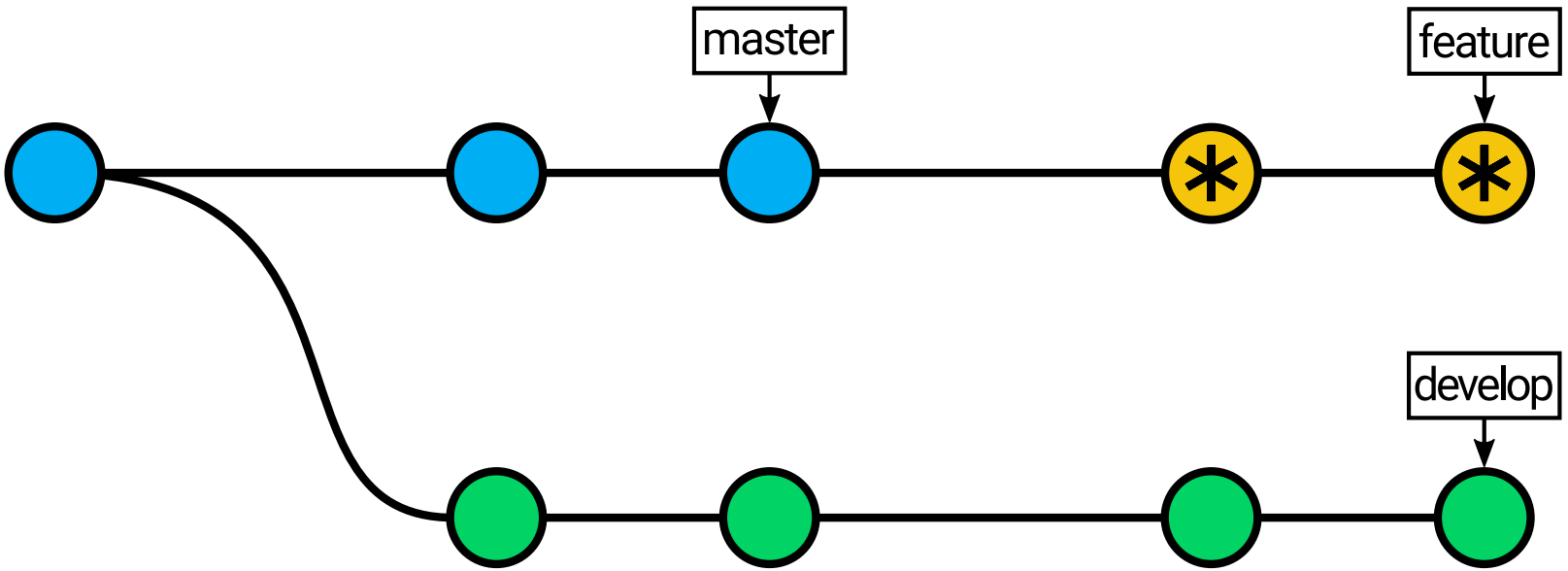
`git rebase --onto <newbase> <oldbase> <branch>`:
Verplaats de `branch` branch **van** `oldbase` **naar** `newbase`.



Complexere rebase

ULYSSIS

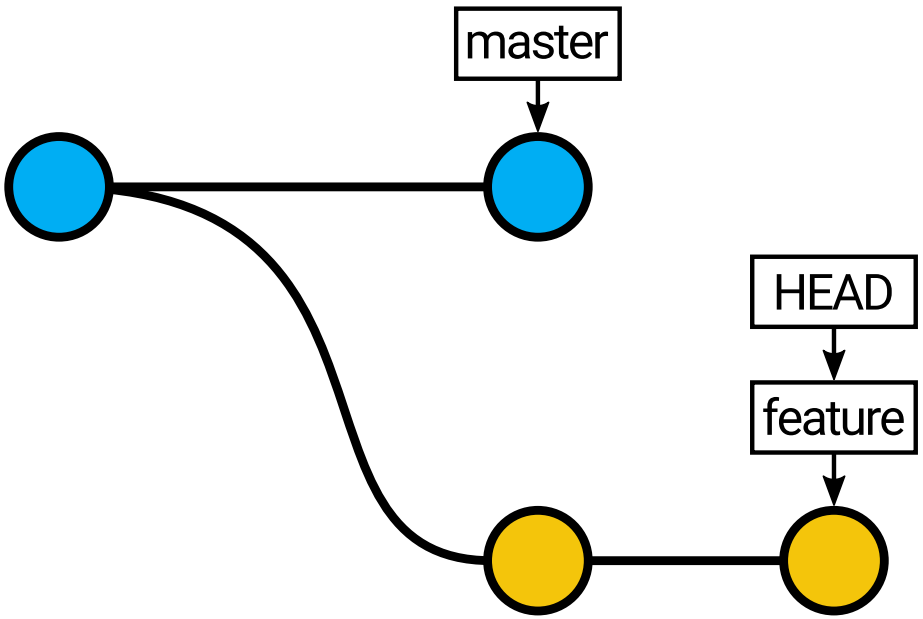
`git rebase --onto <newbase> <oldbase> <branch>`:
Verplaats de `branch` branch **van** `oldbase` **naar** `newbase`.



Gevaren van rebase

ULYSSIS

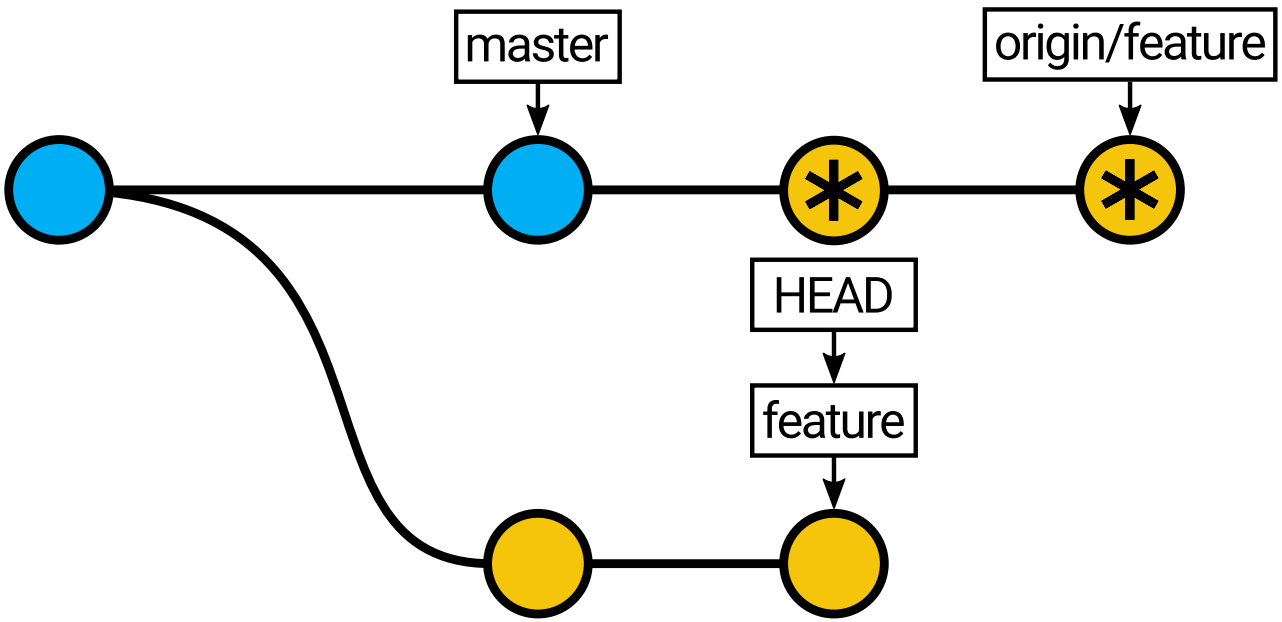
Rebase **enkel** lokale commits!



Gevaren van rebase

ULYSSIS

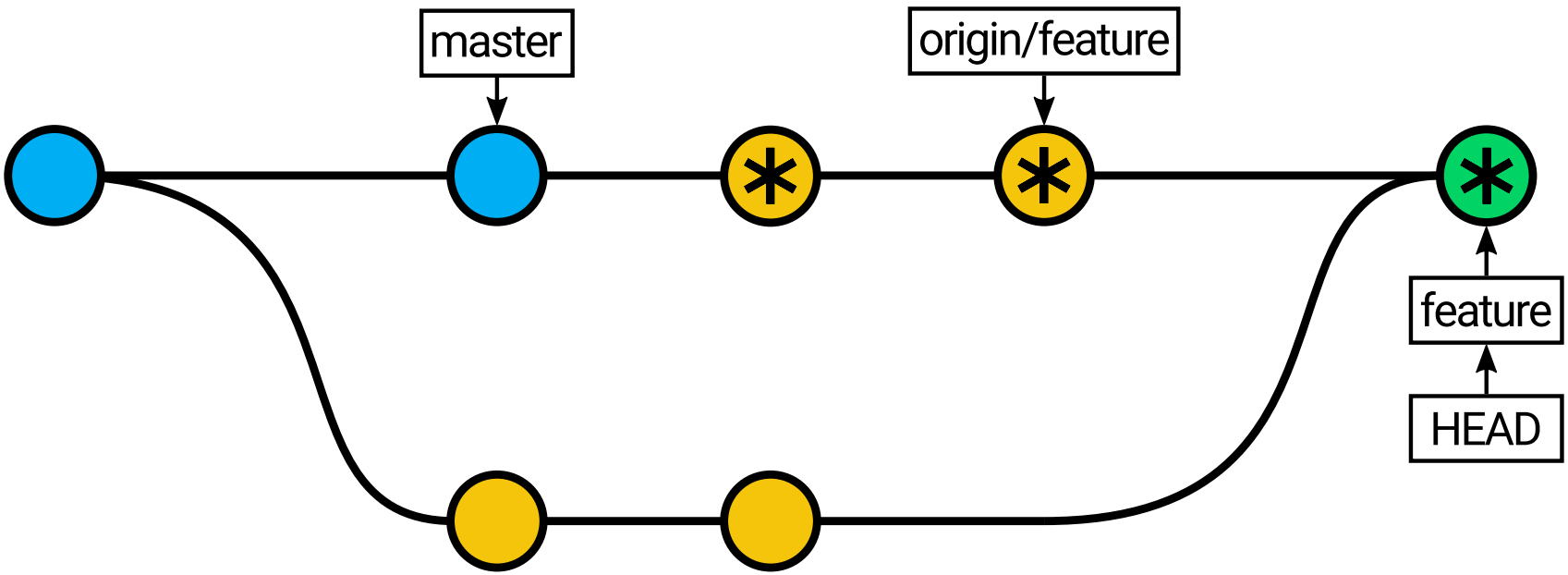
Rebase **enkel** lokale commits!



Gevaren van rebase

ULYSSIS

Rebase **enkel** lokale commits!



Veilig rebasen

Als je toch remote wilt rebasen moet je “force-pushen”.

- Gebruik `--force-with-lease` i.p.v. `--force`
  - Doet enkel een force push als de remote branch niet veranderd is.
- Verwijder de branch na rebase en fast-forward.
  - Want originele branch is niet meer nodig.
  - Voorkomt onbedoelde merges bij pull.

Rebasen bij pulls

ULYSSIS

- Gelijktijdige commits op dezelfde branch = veel nutteloze merge commits.
- `git pull --rebase`: rebase lokale commits i.p.v. ze te mergen.
- Of: `git config pull.rebase true` om dit by default te doen!

Conflicts

ULYSSIS

- `git rebase` kan conflicts veroorzaken!
  - Net zoals bij `git merge`
- Oplossen is net zoals bij `git merge`

  `git rebase --continue` nadat conflicts opgelost zijn.
  `git rebase --abort` om te stoppen met rebasen.
- Enig verschil met `git merge`: een rebase kan **meerdere keren** conflicts veroorzaken!

git reflog: commits terughalen

- Rebasen, amenden, . . . , herschrijven commits.

git reflog: commits terughalen

- Rebasen, amenden, . . . , herschrijven commits.
- Oude versies van commits blijven (een tijdje) bestaan!

git reflog: commits terughalen

- Rebasen, amenden, . . . , herschrijven commits.
- Oude versies van commits blijven (een tijdje) bestaan!
- Maar niet meer op een branch, dus niet meer toegankelijk.

`git reflog`: commits terughalen

- Rebasen, amenden, . . . , herschrijven commits.
- Oude versies van commits blijven (een tijdje) bestaan!
- Maar niet meer op een branch, dus niet meer toegankelijk.

`git reflog` toont alle commits waar `HEAD` heeft gestaan, *inclusief* commits die niet meer toegankelijk zijn!

`git reflog`: commits terughalen

- Rebasen, amenden, . . . , herschrijven commits.
- Oude versies van commits blijven (een tijdje) bestaan!
- Maar niet meer op een branch, dus niet meer toegankelijk.

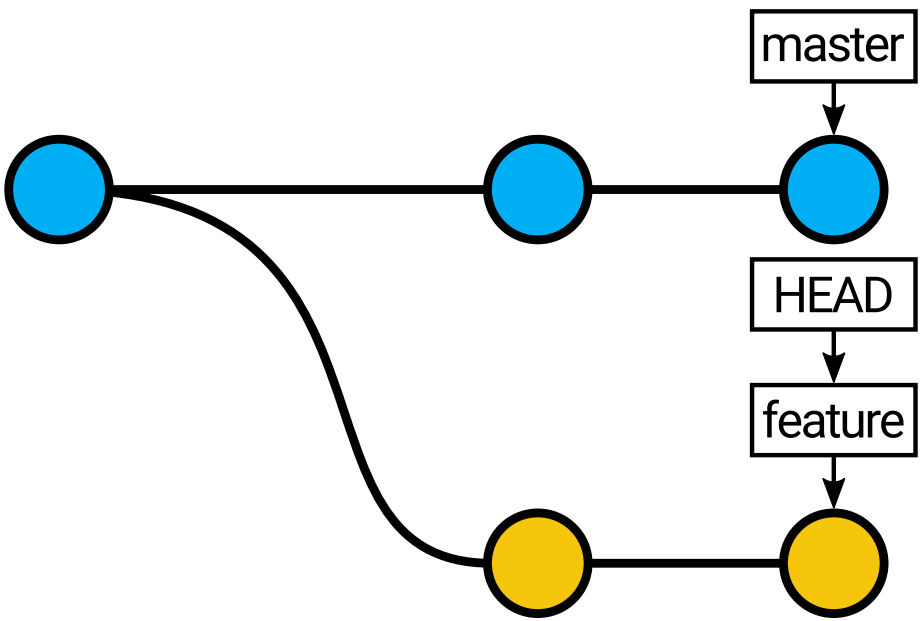
`git reflog` toont alle commits waar `HEAD` heeft gestaan, *inclusief* commits die niet meer toegankelijk zijn!

`git reflog`: commits terughalen

ULYSSIS

- Rebasen, amenden, . . . , herschrijven commits.
- Oude versies van commits blijven (een tijdje) bestaan!
- Maar niet meer op een branch, dus niet meer toegankelijk.

`git reflog` toont alle commits waar `HEAD` heeft gestaan, *inclusief* commits die niet meer toegankelijk zijn!

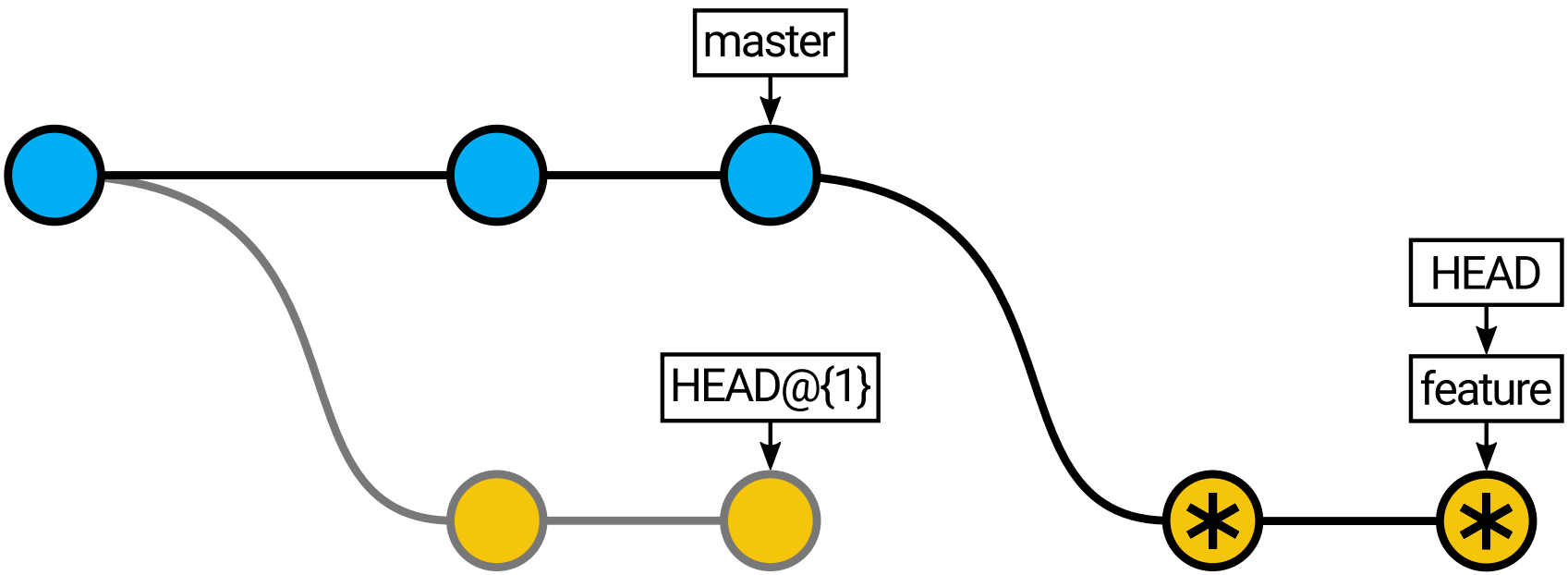


`git reflog`: commits terughalen

ULYSSIS

- Rebasen, amenden, . . . , herschrijven commits.
- Oude versies van commits blijven (een tijdje) bestaan!
- Maar niet meer op een branch, dus niet meer toegankelijk.

`git reflog` toont alle commits waar `HEAD` heeft gestaan, *inclusief* commits die niet meer toegankelijk zijn!



Bugs opsporen met `git bisect`

Hoe kunnen we Git gebruiken om bugs op te sporen?

Bugs opsporen met `git bisect`

Hoe kunnen we Git gebruiken om bugs op te sporen?

- Commit vinden die een bug veroorzaakt heeft.

Bugs opsporen met `git bisect`

ULYSSIS

Hoe kunnen we Git gebruiken om bugs op te sporen?

- Commit vinden die een bug veroorzaakt heeft.
- Te testen commits beperken met binary search!

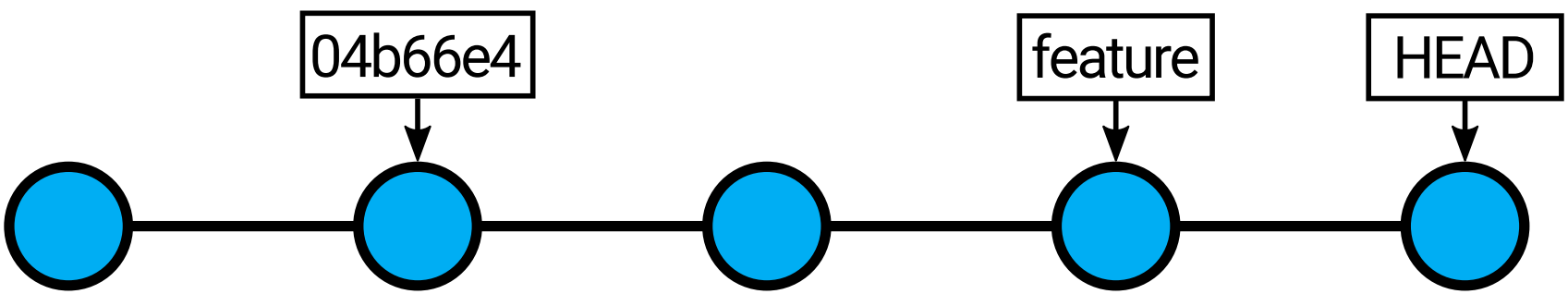
git bisect workflow

1. Start de bisect:
   1.1 `git bisect start`: Start een bisect.
   1.2 `git bisect bad`: Markeer de huidige commit als bad.
   1.3 `git bisect good v1.2`: Markeer een known-good commit.
2. `git bisect` checkout een commit in het midden.
3. Test deze commit:
   - De bug is er niet: `git bisect good`
   - De bug is er wel: `git bisect bad`
   - De commit is slecht om te testen: `git bisect skip`
4. Herhaal tot er geen commits meer zijn.
5. `git bisect` print de eerste commit met de bug.
6. `git bisect reset` brengt je terug waar je was.

Tilde

ULYSSIS

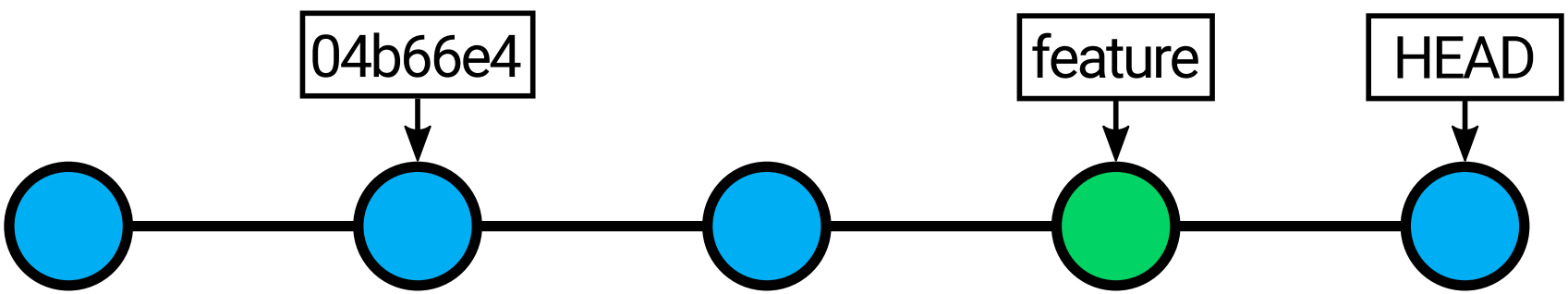
~ na een commit betekent “de parent van deze commit”.



Tilde

~ na een commit betekent “de parent van deze commit”.

`HEAD~` de parent van `HEAD`



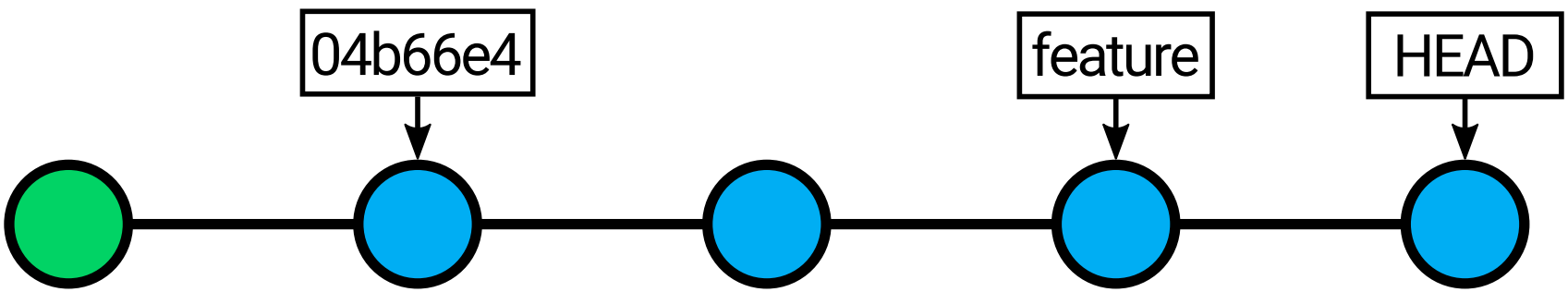
Tilde

ULYSSIS

~ na een commit betekent “de parent van deze commit”.

`HEAD~` de parent van `HEAD`

`04b66e4~` de parent van `04b66e4`

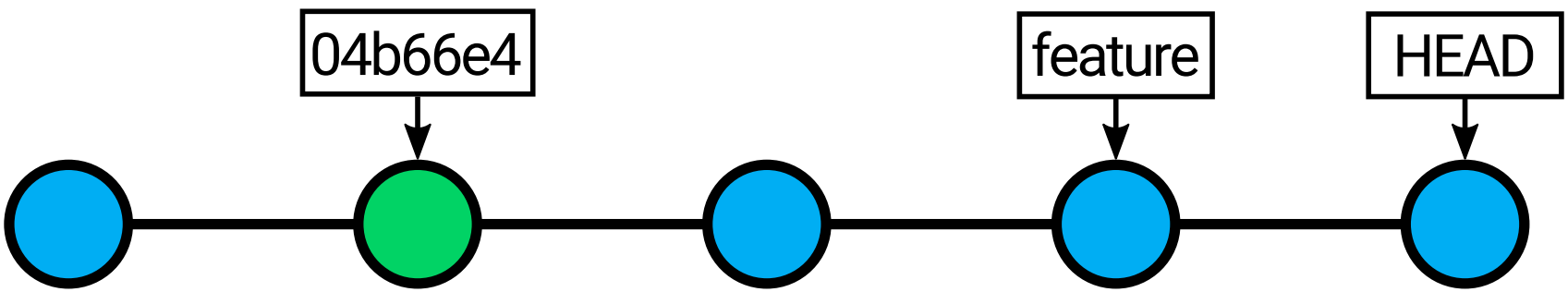


Tilde

ULYSSIS

~ na een commit betekent “de parent van deze commit”.

`HEAD~`	de parent van `HEAD`
`04b66e4~`	de parent van `04b66e4`
`feature~~`	de parent van de parent van `feature`

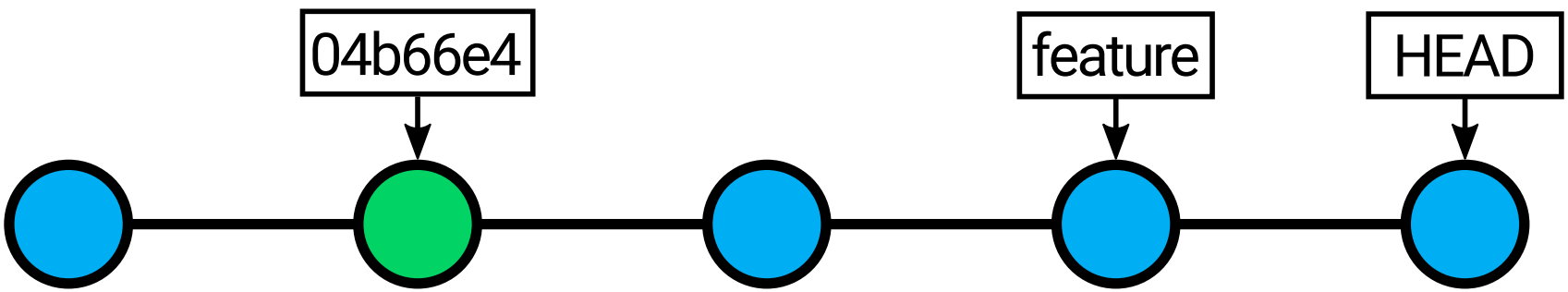


Tilde

ULYSSIS

~ na een commit betekent “de parent van deze commit”.

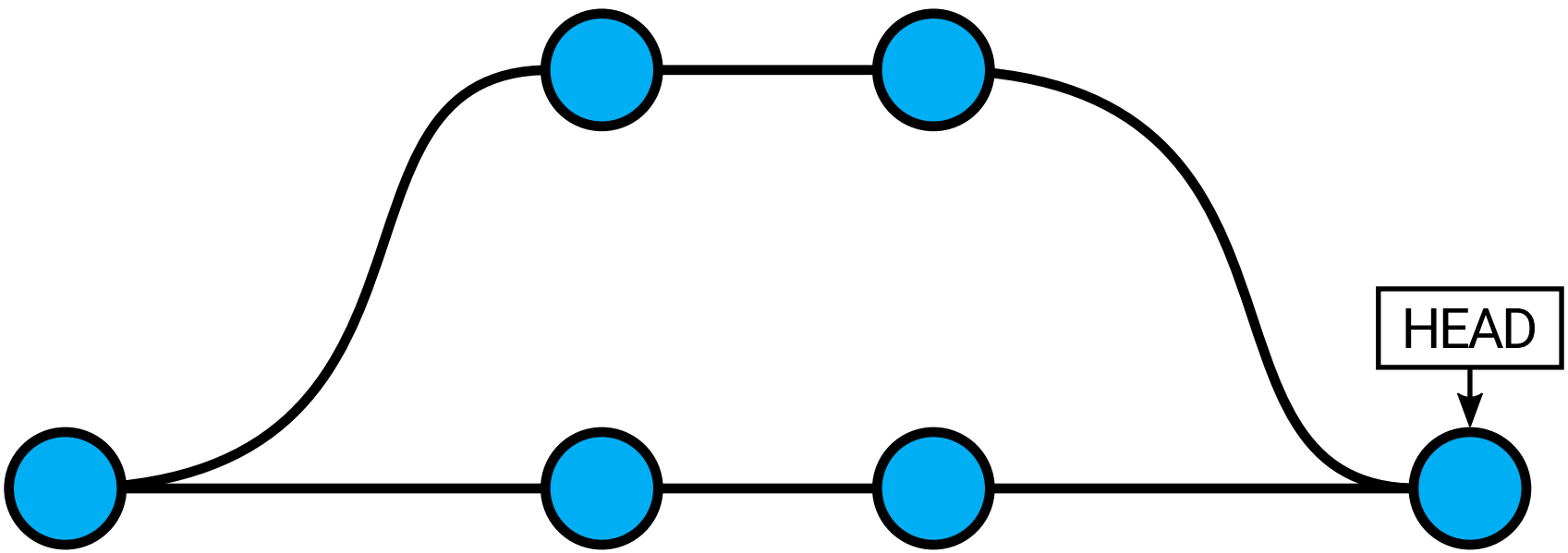
`HEAD~`	de parent van `HEAD`
`04b66e4~`	de parent van `04b66e4`
`feature~~`	de parent van de parent van `feature`
`feature~2`	hetzelfde!



Caret

ULYSSIS

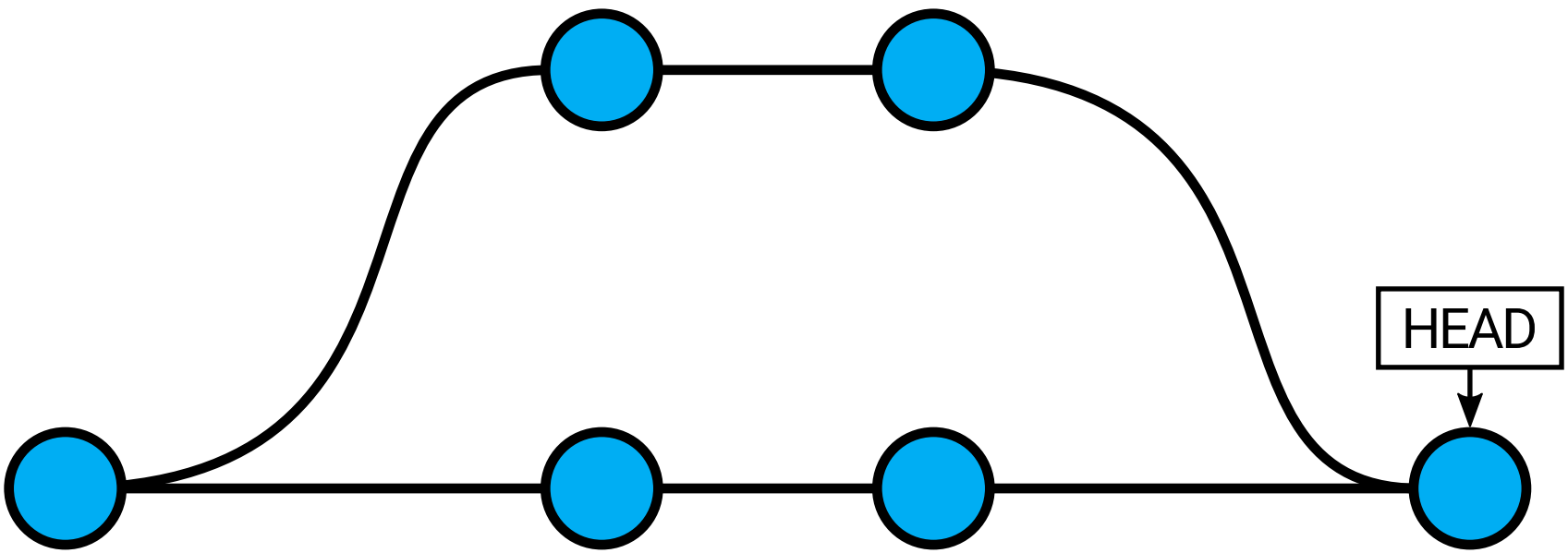
^ betekent “de **eerste** parent van deze commit”.



Caret

ULYSSIS

^ betekent “de **eerste** parent van deze commit”.
^2 betekent “de **tweede** parent van deze commit”.

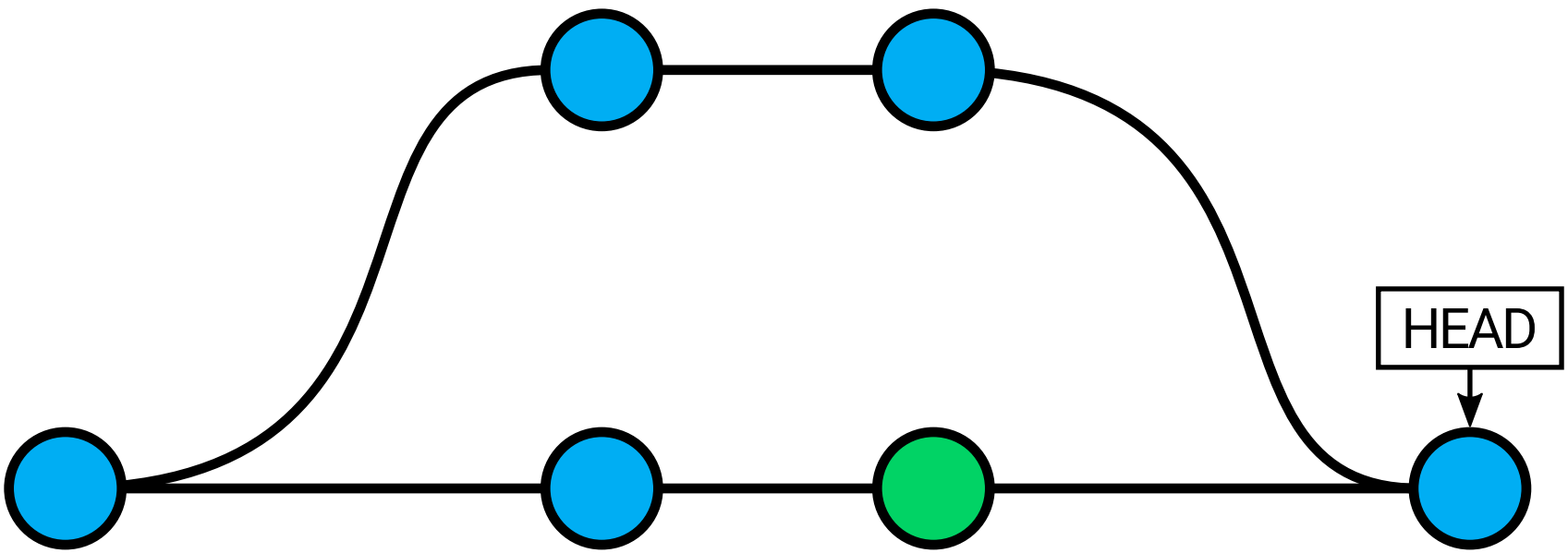


Caret

ULYSSIS

^ betekent “de **eerste** parent van deze commit”.
^2 betekent “de **tweede** parent van deze commit”.

`HEAD^` hetzelfde als `HEAD~`



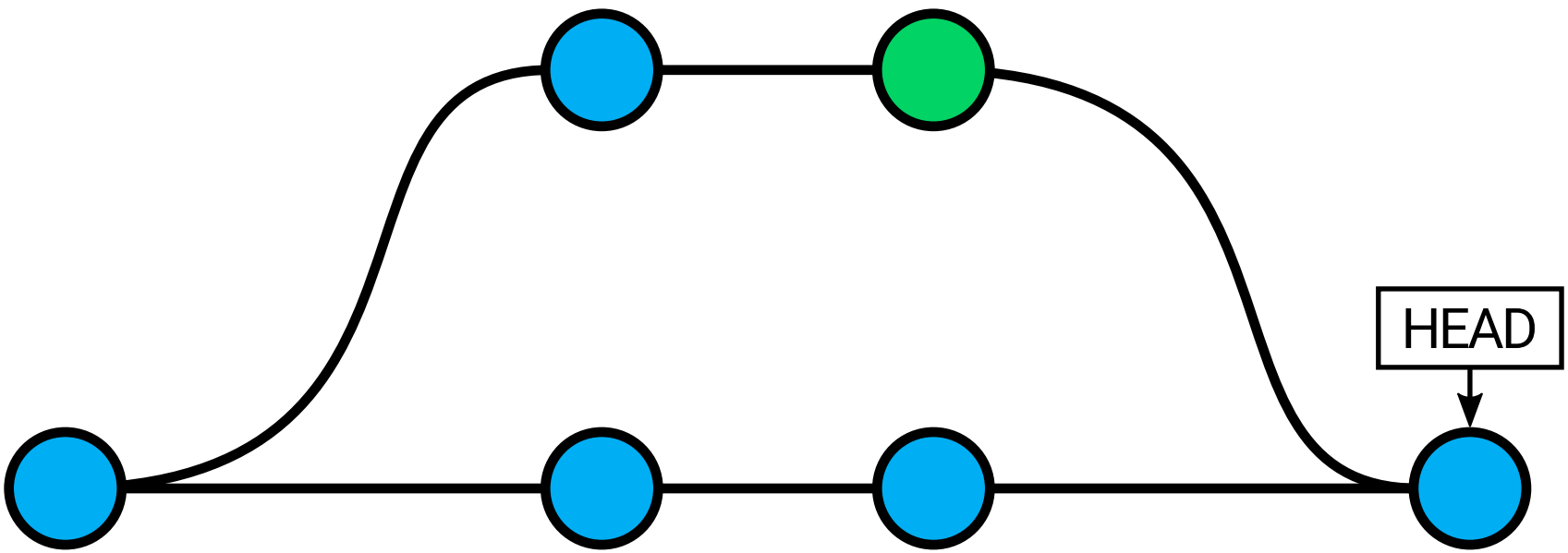
Caret

ULYSSIS

^ betekent “de **eerste** parent van deze commit”.
^2 betekent “de **tweede** parent van deze commit”.

`HEAD^` hetzelfde als `HEAD~`

`HEAD^2` de andere parent van `HEAD`

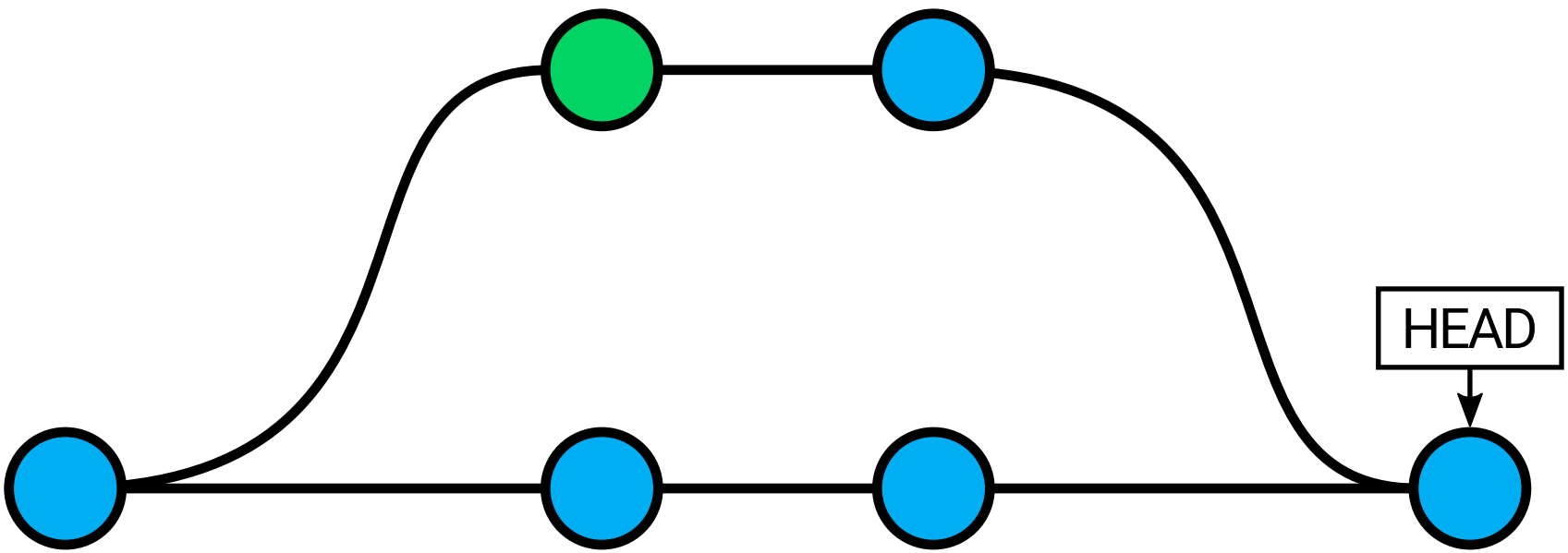


Caret

ULYSSIS

^ betekent "de **eerste** parent van deze commit".
^2 betekent "de **tweede** parent van deze commit".

`HEAD^`	hetzelfde als `HEAD~`
`HEAD^2`	de andere parent van `HEAD`
`HEAD^2~`	de parent van de 2e parent van `HEAD`

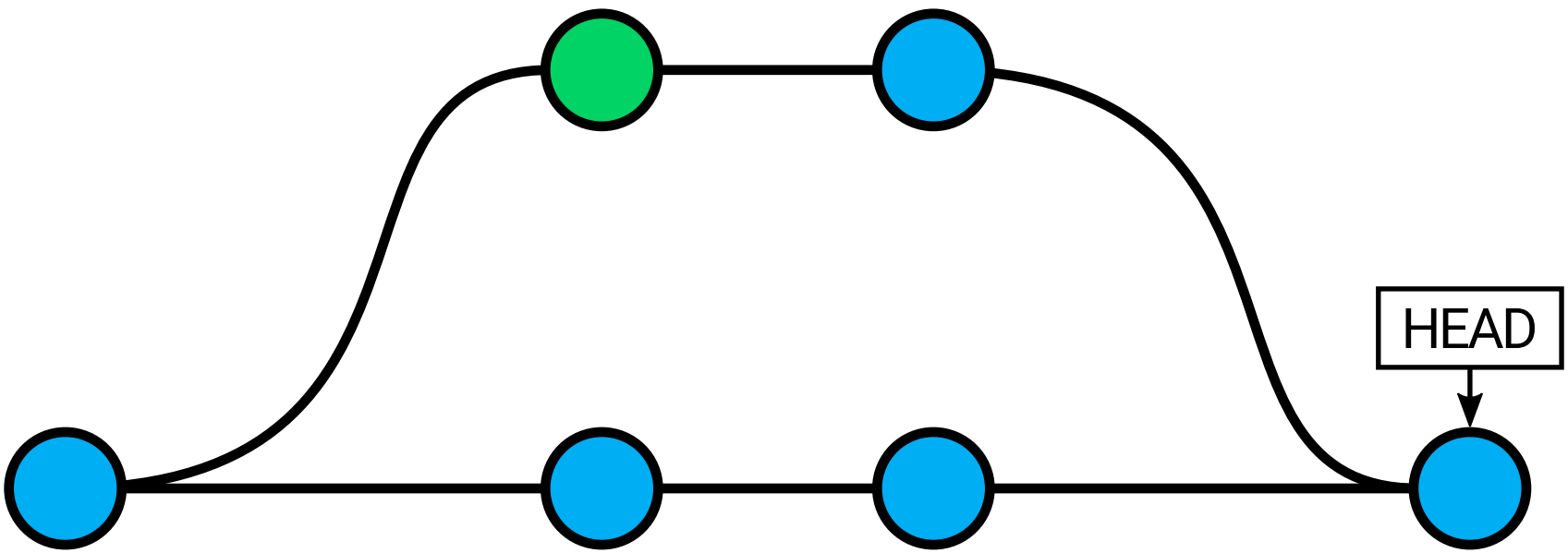


Caret

ULYSSIS

^ betekent "de **eerste** parent van deze commit".
^2 betekent "de **tweede** parent van deze commit".

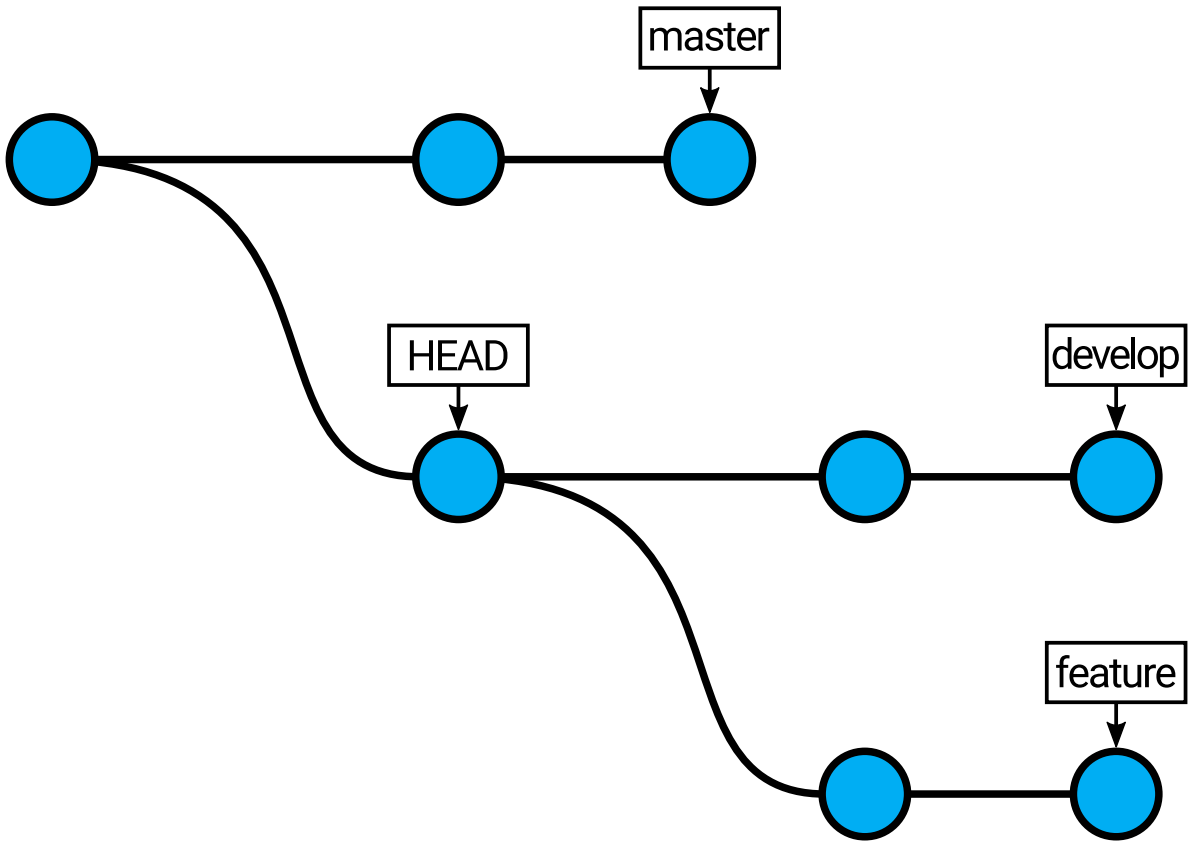
`HEAD^`	hetzelfde als `HEAD~`
`HEAD^2`	de andere parent van `HEAD`
`HEAD^2~`	de parent van de 2e parent van `HEAD`



Commit ranges

ULYSSIS

Sommige commando's, zoals `git log`, kan je een *bereik* van commits meegeven.

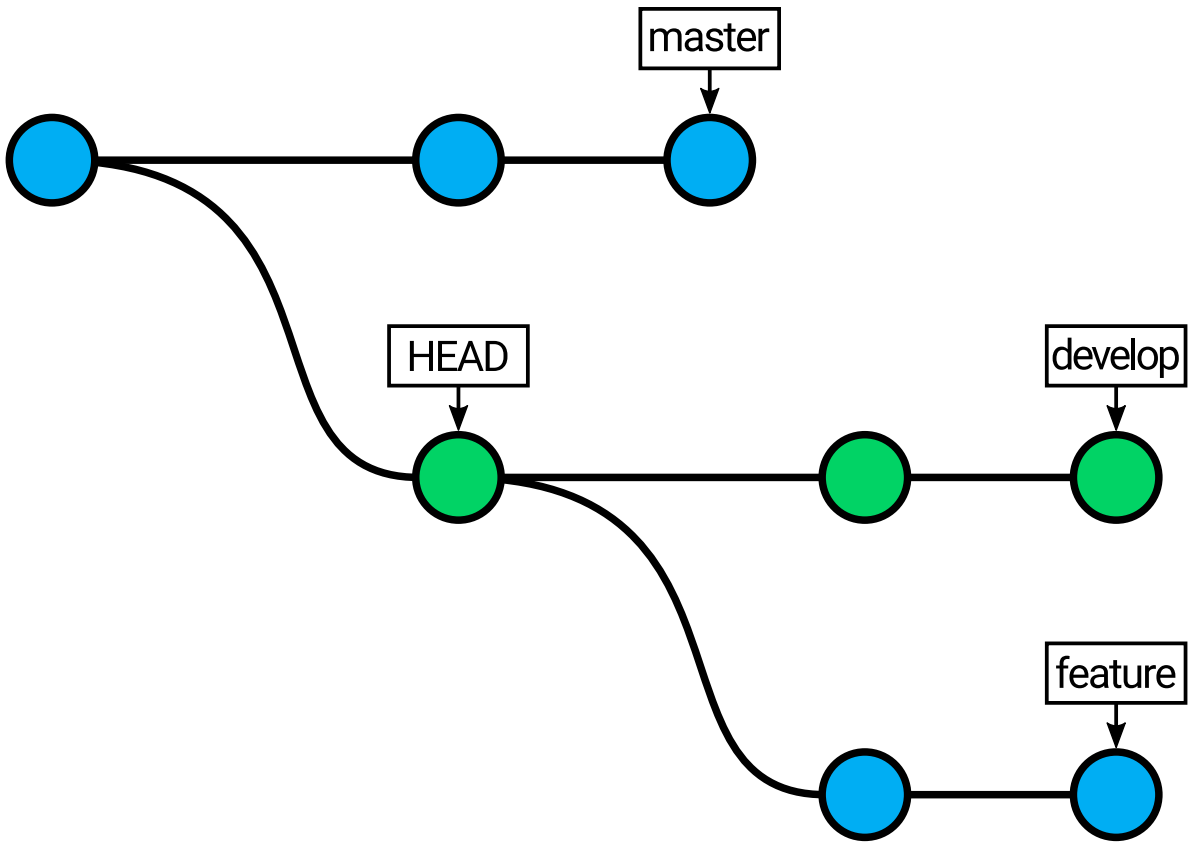


Commit ranges

ULYSSIS

`git log develop --not master`: alle commits die in `develop` maar niet in `master` zitten.

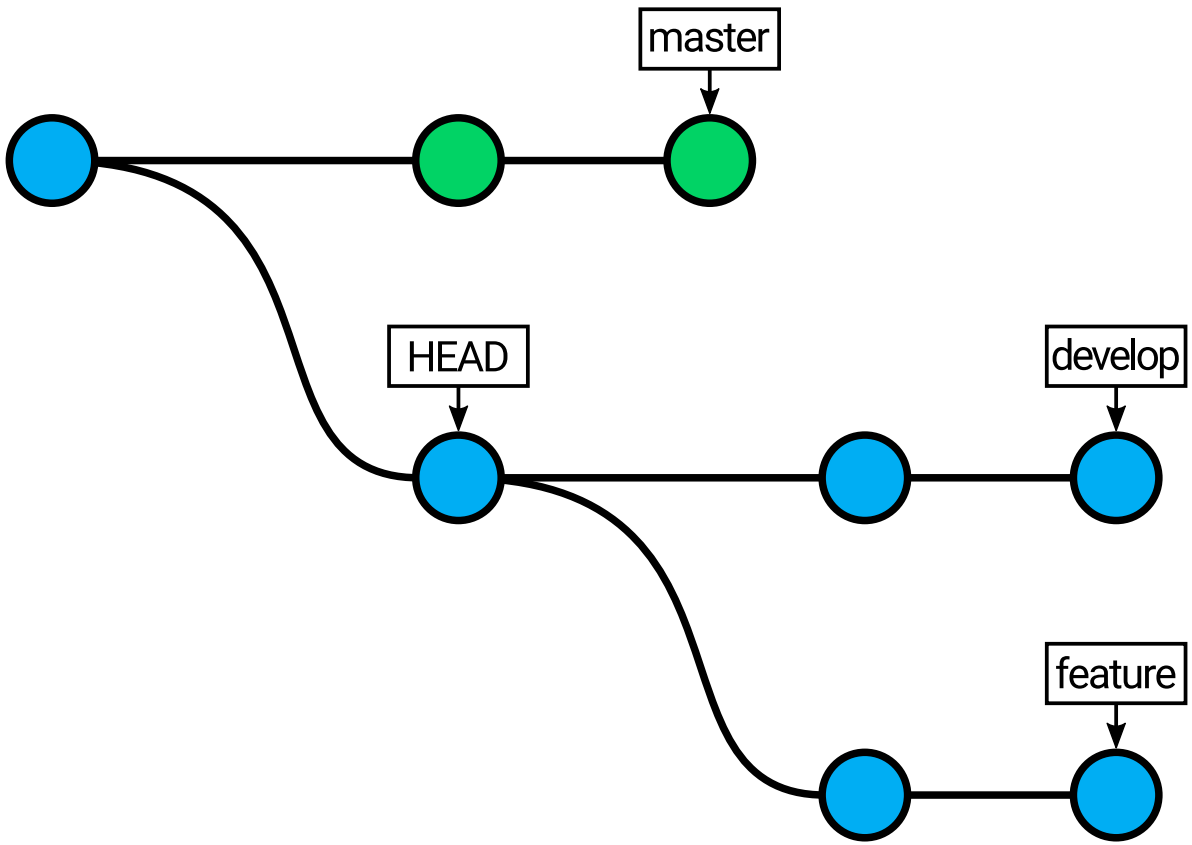
- Kortere syntax: `git log master..develop`



Commit ranges

ULYSSIS

Omgekeerd gaat ook: `git log develop..master` geeft alle commits in `master` nadat `develop` gebrancht is.

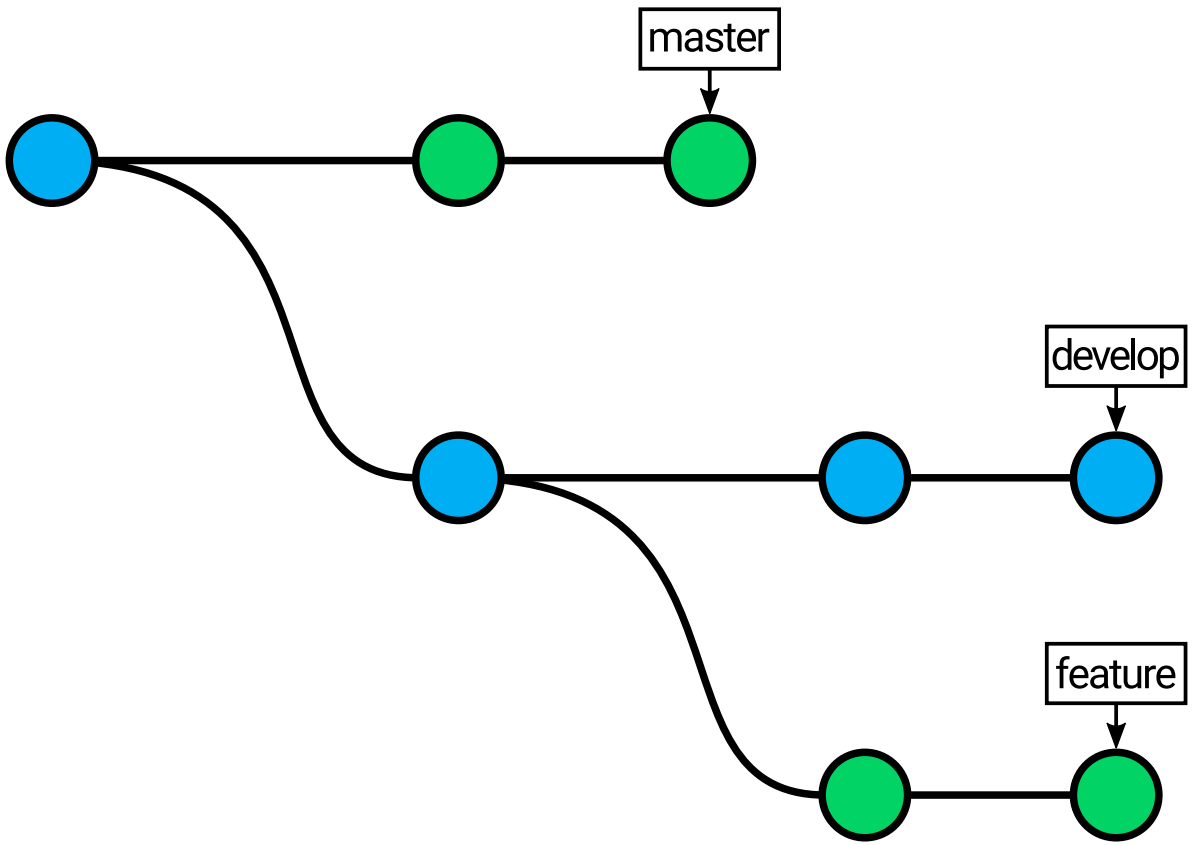


Commit ranges

ULYSSIS

Je kan meer dan 2 refs opgeven:

```
git log master feature --not develop
```



ULYSSIS

git help

gebruik het.